



**RPC BROKER**

**GETTING STARTED WITH THE  
BROKER DEVELOPMENT KIT (BDK)**

Version 1.1

Revised May 2002

Department of Veterans Affairs  
**VISTA** System Design & Development (SD&D)  
Information Infrastructure Service (IIS)



# Document Revision History

The following table displays the revision history for this document. Revisions to the documentation are based on patches and new versions released to the field.

Date	Revision	Description	Author
05/08/02	3.0	Revised Version for Patch 26.	Thom Blom, Oakland OIFO
05/01/02	2.0	Revised Version for Patch 13.	Thom Blom, Oakland OIFO
09/97	1.0	Initial RPC Broker Version 1.1 software release.	Thom Blom, San Francisco OIFO



For a complete list of patches released with the RPC Broker V. 1.1 software, please refer to "Appendix A—Patch Revision History."



# Contents

Orientation .....	ix
How to Use this Manual .....	ix
Commonly Used Terms.....	x
How to Obtain Technical Information Online.....	x
Assumptions About the Reader .....	xi
Reference Materials.....	xii
<b>1. Introduction .....</b>	<b>1-1</b>
<b>2. TRPCBroker Component for Delphi.....</b>	<b>2-1</b>
TRPCBroker Component Properties and Methods .....	2-1
TRPCBroker Key Properties .....	2-2
TRPCBroker Key Methods .....	2-3
How to Connect to an M Server.....	2-4
<b>3. Remote Procedure Calls (RPCs) .....</b>	<b>3-1</b>
What is a Remote Procedure Call? .....	3-1
Create Your Own RPCs .....	3-1
Writing M Entry Points for RPCs .....	3-2
RPC Entry in the REMOTE PROCEDURE File .....	3-5
What Makes a Good Remote Procedure Call? .....	3-5
How to Execute an RPC from a Client Application.....	3-5
RPC Security: How to Register an RPC.....	3-6
<b>4. Other RPC Broker APIs.....</b>	<b>4-1</b>
GetServerInfo Function.....	4-1
VISTA Splash Screen Procedures .....	4-2
XWB GET VARIABLE VALUE RPC.....	4-3
M Emulation Functions .....	4-3
Encryption Functions .....	4-4
\$\$BROKER^XWBLIB .....	4-4
\$\$RTRNFMT^XWBLIB.....	4-4

<b>5. Debugging and Troubleshooting.....</b>	<b>5-1</b>
How to Debug Your Client Application.....	5-1
Troubleshooting Connections.....	5-2
<b>6. RPC Broker Developer Utilities.....</b>	<b>6-1</b>
Programmer Settings .....	6-1
<b>7. RPC Broker and Delphi.....</b>	<b>7-1</b>
Delphi 6.0 Packages .....	7-1
Delphi V. 6 Standard Edition <i>Not</i> Recommended for BDK Development.....	7-1
XWB_Rxx.BPL File.....	7-1
Delphi 5.0 Packages .....	7-1
Delphi V. 5 Standard Edition <i>Not</i> Recommended for BDK Development.....	7-1
XWB_Rxx.BPL File.....	7-2
Delphi 4.0 Packages .....	7-2
XWB_Rxx.BPL File.....	7-2
Delphi 3.0 Packages .....	7-2
VistaBroker.DPL .....	7-2
Distributing the Delphi VCL30.DPL .....	7-3
<b>8. RPC Broker Dynamic Link Library (DLL).....</b>	<b>8-1</b>
DLL Interface.....	8-1
Exported Functions.....	8-1
Header Files Provided.....	8-1
Sample DLL Application.....	8-1
Return Values from RPCs.....	8-2
COTS Development and the DLL .....	8-2
<b>9. For More Information.....</b>	<b>9-1</b>
RPC Broker Developer's Guide—BROKER.HLP .....	9-1
Other RPC Broker Resources .....	9-2
Glossary .....	Glossary-1
Appendix A—Patch Revision History .....	Appendix A-1
Index .....	Index-1-1

# Figures

Table 1: Documentation symbol descriptions.....	ix
Table 2: Commonly used RPC Broker terms.....	x
Table 3: TRPCBroker component key properties.....	2-2
Table 4: RPC Broker return value types .....	3-3
Table 5: Input parameter types.....	3-4
Table 6: REMOTE PROCEDURE file key field entries .....	3-5
Figure 1: Server and port configuration selection dialog.....	4-1
Figure 2: VISTA Splash screen .....	4-2
Figure 3: RPC Broker Programmer Preferences dialog.....	6-1
Table 7: Programmer preference settings .....	6-1
Table 8: TRPCBroker component's Results property .....	8-2
Figure 4: Delphi's Tool Properties dialog .....	9-1
Table 9: Related documentation (and format) .....	9-2
Table 10: RPC Broker V. 1.1 patch revision history (in reverse sequence order) .....	Appendix-7





# Orientation



## How to Use this Manual

Throughout this manual, advice and instructions are offered regarding the use of the RPC Broker V. 1.1 Broker Development Kit (BDK) and the functionality it provides for Veterans Health Information Systems and Technology Architecture (VISTA) and commercial off-the-shelf (COTS) software products.

There are no special legal requirements involved in the use of the RPC Broker's Interface.

This manual uses several methods to highlight different aspects of the material:

- Various symbols are used throughout the documentation to alert the reader to special information. The following table gives a description of each of these symbols:

Symbol	Description
	Used to inform the reader of general information including references to additional reading material
	Used to caution the reader to take special notice of critical information

**Table 1: Documentation symbol descriptions**

- Descriptive text is presented in a proportional font (as represented by this font).
- "Snapshots" of computer online displays (i.e., roll-and-scroll screen captures/dialogs) and computer source code are shown in a *non*-proportional font and enclosed within a box. Also included are Graphical User Interface (GUI) Microsoft Windows images (i.e., dialogs or forms).
  - User's responses to online prompts will be boldface type.
  - The "<Enter>" found within these snapshots indicate that the user should press the Enter or Return key on their keyboard.
  - Author's comments are displayed in italics or as "callout" boxes.




Callout boxes refer to labels or descriptions usually enclosed within a box, which point to specific areas of a displayed image.

- Object Pascal code uses a combination of upper- and lowercase characters. All Object Pascal reserved words are in boldface type.
- All uppercase is reserved for the representation of M code, variable names, or the formal name of options, field and file names, and security keys (e.g., the XUPROGMODE key).

## Commonly Used Terms

The following is a list of terms and their descriptions that you may find helpful while reading the RPC Broker documentation:

Term	Description
<b>Client</b>	A single term used interchangeably to refer to a user, the workstation (i.e., PC), and the portion of the program that runs on the workstation.
<b>Component</b>	<p>A software object that contains data and code. A component may or may not be visible.</p> <p> For a more detailed description, see the "Borland Delphi for Windows User Guide."</p>
<b>GUI</b>	The Graphical User Interface application that is developed for the client workstation.
<b>Host</b>	The term Host is used interchangeably with the term Server.
<b>Server</b>	The computer where the data and the RPC Broker remote procedure calls (RPCs) reside.

**Table 2: Commonly used RPC Broker terms**



Please refer to the "Glossary" for additional terms and definitions.

## How to Obtain Technical Information Online

Exported file, routine, and global documentation can be generated through the use of Kernel, MailMan, and VA FileMan utilities.



Methods of obtaining specific technical information online will be indicated where applicable under the appropriate topic.

### Help at Prompts

**VISTA** software has online help and commonly used system default prompts. In roll-and-scroll mode users are strongly encouraged to enter question marks at any response prompt. At the end of the help display, you are immediately returned to the point from which you started. This is an easy way to learn about any aspect of **VISTA** software.

To retrieve online documentation in the form of Help in **VISTA** roll-and-scroll software:

- Enter a single question mark ("?",) at a field/prompt to obtain a brief description. If a field is a pointer, entering one question mark ("?",) displays the HELP PROMPT field contents and a list of choices, if the list is short. If the list is long, the user will be asked if the entire list should be displayed. A YES response will invoke the display. The display can be given a starting point by prefacing the starting point with an up-arrow ("^") as a response. For example, ^**M** would start an alphabetic listing at the letter M instead of the letter A while ^**127** would start any listing at the 127th entry.
- Enter two question marks ("??") at a field/prompt for a more detailed description. Also, if a field is a pointer, entering two question marks displays the HELP PROMPT field contents and the list of choices.
- Enter three question marks ("???",) at a field/prompt to invoke any additional Help text that may be stored in Help Frames.

## Obtaining Data Dictionary Listings

Technical information about files and the fields in files is stored in data dictionaries. You can use the List File Attributes option on the Data Dictionary Utilities submenu in VA FileMan to print formatted data dictionaries.



For details about obtaining data dictionaries and about the formats available, please refer to the "List File Attributes" chapter in the "File Management" section of the "VA FileMan Advanced User Manual."

## Assumptions About the Reader

This manual is written with the assumption that the reader is familiar with the following:

- **VISTA** computing environment (e.g., Kernel Installation and Distribution System [KIDS])
- VA FileMan data structures and terminology
- Microsoft Windows
- Borland's Delphi development environment
- GUI standards and guidelines
- M programming language
- Object Pascal programming language

No attempt is made to explain how the overall **VISTA** programming system is integrated and maintained. Such methods and procedures are documented elsewhere. We suggest you look at the various VA home pages on the World Wide Web for a general orientation to **VISTA**. For example, go to the System Design & Development (SD&D) Home Page at the following web address:

<http://vista.med.va.gov/>

This manual does provide, however, an explanation of the RPC Broker, describing how it can be used in a client/server environment.

## Reference Materials

Readers who wish to learn more about the RPC Broker should consult the following:

- "RPC Broker Developer's Guide" (i.e., BROKER.HLP, online help designed for programmers, distributed in the BDK)
- "RPC Broker Systems Manual"
- "RPC Broker Technical Manual"
- "RPC Broker Installation Guide"
- "RPC Broker Release Notes"
- RPC Broker Home Page at the following web address:

<http://vista.med.va.gov/broker/>

This site provides announcements, additional information (e.g., Frequently Asked Questions [FAQs], advisories), documentation links, archives of older documentation and software downloads.

Broker documentation is made available online, on paper, and in Adobe Acrobat Portable Document Format (.PDF). The .PDF documents must be read using the Adobe Acrobat Reader (i.e., ACROREAD.EXE), which is freely distributed by Adobe Systems Incorporated at the following web address:

<http://www.adobe.com/>



For more information on the use of the Adobe Acrobat Reader, please refer to the "Adobe Acrobat Quick Guide" at the following web address:

<http://vista.med.va.gov/iis/acrobat/index.html>



**DISCLAIMER: The appearance of external hyperlink references in this manual does not constitute endorsement by the Department of Veterans Health Administration (VHA) of this Web site or the information, products, or services contained therein. The VHA does not exercise any editorial control over the information you may find at these locations. Such links are provided and are consistent with the stated purpose of this VHA Intranet Service.**

# 1. Introduction

The Remote Procedure Call (RPC) Broker (also referred to as "Broker") is a client/server system within VA's Veterans Health Information Systems and Technology Architecture (**VISTA**) environment. It establishes a common and consistent foundation for client/server applications being written as part of **VISTA**. It enables client applications to communicate and exchange data with M Servers.

This manual introduces developers to the RPC Broker and the Broker Development Kit (BDK). The emphasis is on using the RPC Broker in conjunction with Borland's Delphi software. However, the RPC Broker supports other development environments.

This manual provides an overview of development with the RPC Broker.



For more complete information on development with the RPC Broker components, please refer to the "RPC Broker Developer's Guide" (i.e., BROKER.HLP, online help distributed with the BDK).

This document is intended for the **VISTA** development community and Information Resource Management (IRM) staff. A wider audience of technical personnel engaged in operating and maintaining the Department of Veterans Affairs (VA) software may also find it useful as a reference.

## About This Version of the BDK

This version of the BDK provides programmers with the capability to develop new **VISTA** client/server software using the Broker Delphi component (i.e., TRPCBroker) in a 32-bit environment.

To develop **VISTA** applications in a 32-bit environment you must have Delphi V. 2.0 or greater. This version of the RPC Broker component will *not* allow you to develop applications in Delphi V. 1.0. However, the Broker routines on the M server will continue to support **VISTA** applications previously developed in the 16-bit environment.

The default installation of the Broker creates a separate BDK directory (i.e., BDK32) that contains the required Broker files for development.

## Backward Compatibility Issues

Client applications compiled with this version of the RPC Broker (V. 1.1) will not work at a site that has not upgraded its RPC Broker server software to V.1.1.

On the other hand, client applications compiled with RPC Broker V.1.0 will work with the V. 1.1 RPC Broker server.



## 2. TRPCBroker Component for Delphi

The main tool to develop client applications for the RPC Broker environment is the TRPCBroker component for Delphi. The TRPCBroker component adds the following abilities to your Delphi application:

- **Connecting to an M server**
  - Authenticate the user
  - Set up the environment on the server
  - Bring back the introductory text
- **Invoking Remote Procedure Calls (RPCs) on the M Server**
  - Send data to the M Server
  - Perform actions on the server
  - Return data from the server to the client

To add the TRPCBroker component to your Delphi application, simply drop it from the Kernel tab of Delphi's component palette to a form in your application.

### TRPCBroker Component Properties and Methods

As a Delphi component, the TRPCBroker component is controlled and accessed through its properties and methods. By setting its properties and executing its methods, you can connect to an M server from your application and execute RPCs on the M server to exchange data and perform actions on the M server.

For most applications, you will only need to use a single TRPCBroker component to manage communications with the M server.

## TRPCBroker Key Properties

The following table lists the most important properties of the TRPCBroker component.



For a complete list of all of Broker properties, please refer to the "RPC Broker Developer's Guide" (i.e., BROKER.HLP, online help distributed with the BDK).

Property	Description
<b>ClearParameters</b>	If True, the Param property is cleared <i>after</i> every invocation of the Call, strCall, or the lstCall methods.
<b>ClearResults</b>	If True, the Results property is cleared <i>before</i> every invocation of the Call method, thus assuring that only the results of the last call are returned.
<b>Connected</b>	Setting this property to True connects your application to the server.
<b>ListenerPort</b>	Sets server port to connect to a Broker Listener process (mainly for development purposes; for end-users, determine on the fly with GetServerInfo method.)
<b>Param</b>	Run-time array in which you set any parameters to pass as input parameters when calling an RPC on the server.
<b>RemoteProcedure</b>	Name of a RemoteProcedure entry that the Call, lstCall, or strCall method should invoke.
<b>Results</b>	This is where any results are stored after a Call, lstCall, or strCall method completes.
<b>Server</b>	Name of the server to connect to (mainly for development purposes; for end-users, determine on the fly with GetServerInfo method.)

**Table 3: TRPCBroker component key properties**



## TRPCBroker Key Methods

This section lists the most important methods of the TRPCBroker component.



For a complete list of all of Broker methods, please refer to the "RPC Broker Developer's Guide" (i.e., BROKER.HLP, online help distributed with the BDK).

### **procedure Call;**

This method executes an RPC on the server and returns the results in the TRPCBroker component's Results property.

Call expects the name of the remote procedure and its parameters to be set up in the RemoteProcedure and Param properties respectively. If ClearResults is True, then the Results property is cleared before the call. If ClearParameters is True, then the Param property is cleared after the call finishes.

### **function strCall: string;**

This method is a variation of the Call method. Only use it when the return type is a single string. Instead of returning results in the TRPCBroker component's Results[0] property node, results are returned as the value of the function call. Unlike the Call method, the Results property is not affected; no matter the setting of ClearResults, the value is left unchanged.

### **procedure lstCall(OutputBuffer: TStrings);**

This method is a variation of the Call method. Instead of returning results in the TRPCBroker component's Results property, it instead returns results in the TStrings object you specify. Unlike the Call method, the Results property is not affected; no matter the setting of ClearResults, the value is left unchanged.

### **function CreateContext(strContext: string): boolean;**

This method creates a context for your application. Pass an option name in the strContext parameter. If the function returns True, a context was created, and your application can use all RPCs entered in the option's RPC multiple.

## **Examples**

For examples of how to use these methods to invoke RPCs, see the "How to Execute an RPC from a Client Application" topic in the "Remote Procedure Calls" chapter of this manual.

## How to Connect to an M Server

To establish a connection from your application to a Broker server:

1. From the Kernel component palette tab, add a TRPCBroker component to your form.
2. Add code to your application to connect to the server; one likely location is your form's OnCreate event handler. The code should:
  - a. Use the GetServerInfo function to retrieve the run-time server and port to connect to. This function is not a method of the TRPCBroker component; it is described in the Other RPC Broker APIs chapter.
  - b. Inside of an exception handler **try...except** block, set RPCBroker1's Connected property to True. This causes an attempt to connect to the Broker server.
  - c. Check if an EBrokerError exception is raised. If this happens, connection failed. You should inform the user of this and then terminate the application.

The code, placed in an OnCreate event handler, should look like:

```
procedure TForm1.FormCreate(Sender: TObject);
var   ServerStr: String;
      PortStr: String;
begin
  // get the correct port and server from registry
  if GetServerInfo(ServerStr,PortStr)<>mrCancel then
    begin
      RPCBroker1.Server:=ServerStr;
      RPCBroker1.ListenerPort:=StrToInt(PortStr);
    end
    else Application.Terminate;

  // establish a connection to the Broker
  try
    RPCBroker1.Connected:=True;
  except
    On EBrokerError do
      begin
        ShowMessage('Connection to server could not be established!');
        Application.Terminate;
      end;
    end;
  end;
```

3. A connection with the Broker M Server is now established. You can use the CreateContext method of the TRPCBroker component to authorize use of RPCs for your user, and then use the Call, IstCall, and strCall methods of the TRPCBroker component to execute RPCs on the M server. See the next chapter, Remote Procedure Calls, for information on creating and executing RPCs.

## 3. Remote Procedure Calls (RPCs)

### What is a Remote Procedure Call?

A remote procedure call (RPC) is a defined call to M code that runs on an M server. A client application, through the RPC Broker, can make a call to the M server and execute an RPC on the M server. This is the mechanism through which a client application can:

- Send data to an M server
- Execute code on an M server
- Retrieve data from an M server

An RPC can take optional parameters to do some task and then return either a single value or an array to the client application. RPCs are stored in the REMOTE PROCEDURE file (#8994).

### Relationship Between an M Entry Point and an RPC

An RPC can be thought of as a wrapper placed around an M entry point for use with client applications. Each RPC invokes a single M entry point. The RPC passes data in specific ways to its corresponding M entry point and expects any return values from the M entry point to be returned in a pre-determined format. This allows client applications to connect to the RPC Broker, invoke an RPC, and through the RPC, invoke an M entry point on a server.

Delphi  
Component

M Entry  
Point

### Create Your Own RPCs

You can create your own custom RPCs to perform actions on the M server and to retrieve data from the M server. Then you can call these RPCs from your client application. Creating an RPC requires you to perform the following two steps:

1. Write and test the M entry point that is called by the RPC.
2. Add the RPC entry that invokes your M entry point, in the REMOTE PROCEDURE file (#8994).

## Writing M Entry Points for RPCs

### First Input Parameter (Required)

The RPC Broker always passes a variable by reference in the first input parameter to your M routine. It expects results (one of five types described below) to be returned in this parameter. You must always set some return value into that first parameter before your routine returns.

### Return Value Types

There are five RETURN VALUE TYPES for RPCs as shown in the table below. Choose a return value type that is appropriate to the type of data your RPC needs to return to your client. Your M entry point should set the return value (in the routine's first input parameter) accordingly.

RPC Return Value Type	How M Entry Point Should Set the Return Parameter	RPC Word Wrap On Setting	Value(s) returned in Client Results
<b>Single Value</b>	Set the return parameter to a single value. For example: <pre> TAG (RESULT) ; S RESULT="DOE, JOHN" Q </pre>	No effect	Value of parameter, in Results[0].
<b>Array</b>	Set an array of strings into the return parameter, each subscripted one level descendant. For example: <pre> TAG (RESULT) ; S RESULT (1) ="ONE" S RESULT (2) ="TWO" Q </pre> For large arrays consider using the GLOBAL ARRAY return value type to avoid memory allocation errors.	No effect	Array values, each in a Results item.
<b>Word Processing</b>	Set the return parameter the same as you set it for the ARRAY type. The only difference is that the WORD WRAP ON setting affects the WORD PROCESSING return value type.	True	Array values, each in a Results item.
		False	Array values, concatenated into Results[0].


RPC Return Value Type	How M Entry Point Should Set the Return Parameter	RPC Word Wrap On Setting	Value(s) returned in Client Results
<b>Global Array</b>	<p>Set the return parameter to a closed global reference in ^TMP. The global's data nodes will be traversed using \$QUERY, and all data values on global nodes descendant from the global reference are returned.</p> <p>This type is especially useful for returning data from VA FileMan word processing fields, where each line is on a 0-subscripted node.</p> <p> <b>The global reference you pass is killed by the Broker at the end of RPC Execution as part of RPC cleanup. Do not pass a global reference that is not in ^TMP or that should not be killed.</b></p> <p>This type is useful for returning large amounts of data to the client, where using the ARRAY type can exceed the symbol table limit and crash your RPC.</p> <p>For example, to return sign-on introductory text you could do:</p> <pre> TAG (RESULT) ;   M ^TMP ("A6A", \$J) = ^XTV (8989.3, 1, "INTRO") ;this node not needed K ^TMP ("A6A", \$J, 0) S RESULT=\$NA (^TMP ("A6A", \$J) ) Q </pre>	True	Array values, each in a Results item.
		False	Array values, concatenated into Results[0].
<b>Global Instance</b>	<p>Set the return parameter to a closed global reference.</p> <p>For example, to return the 0th node from the NEW PERSON file for the current user:</p> <pre> TAG (RESULT) ; S RESULT=\$NA (^VA (200, DUZ, 0) ) Q </pre>	No effect	Value of global node, in Results[0].

Table 4: RPC Broker return value types

## Input Parameter Types (Optional)

The M entry point for an RPC can optionally have input parameters (i.e., beyond the first parameter, which is always used to return an output value). The client passes data to your M entry point through these parameters.

The client can send data to an RPC (and therefore your entry point) in one of the following three format types:

Param PType	Param Value
<b>Literal</b>	Delphi string value, passed as a string literal to the M server.
<b>Reference</b>	Delphi string value, treated on the M Server as an M variable name and resolved from the symbol table at the time the RPC executes.
<b>List</b>	A single-dimensional array of strings in the Mult subproperty of the Param property, passed to the M Server where it is placed in an array. String subscripting can be used.

**Table 5: Input parameter types**

The type of the input parameters passed in the Param property of the TRPCBroker component determines the format of the data you must be prepared to receive in your M entry point.

## RPC M Entry Point Examples

The following two examples illustrate sample M code that could be used in simple RPCs.

1. This example takes two numbers and returns their sum:

```
SUM (RESULT, A, B)           ;add two numbers
S  RESULT=A+B
Q
```

2. This example receives an array of numbers and returns them as a sorted array to the client:

```
SORT (RESULT, UNSORTED)      ;sort numbers
N I
S  I=""
F  S  I=$O (UNSORTED (I))  Q:I=""  S  RESULT (UNSORTED (I)) =UNSORTED (I)
Q
```

## RPC Entry in the REMOTE PROCEDURE File

After the M code is complete, you need to create the RPC itself in the REMOTE PROCEDURE file (#8994). The following fields in the REMOTE PROCEDURE file (#8994) are key to the correct operation of an RPC:

Field Name	Required?	Description
<b>NAME (#.01)</b>	Yes	The name that identifies the RPC (this entry should be namespaced in the package namespace).
<b>TAG (#.02)</b>	Yes	The tag at which the remote procedure call begins.
<b>ROUTINE (#.03)</b>	Yes	The name of the routine that should be invoked to start the RPC.
<b>WORD WRAP ON (#.08)</b>	No	Affects GLOBAL ARRAY and WORD PROCESSING return value types only. If set to False, data is returned in a single concatenated string in Results[0]. If set to True, each array node on the M side is returned as a distinct array item in Results.
<b>RETURN VALUE TYPE (#.04)</b>	Yes	This indicates to the Broker how to format the return values. For example, if RETURN VALUE TYPE is WORD PROCESSING, then each entry in the returning list will have a <CR><LF> (<carriage return><line feed>) appended.

Table 6: REMOTE PROCEDURE file key field entries

## What Makes a Good Remote Procedure Call?

- Silent calls (no I/O to terminal or screen, no user intervention required)
- Minimal resources required (passes data in brief, controlled increments)
- Discrete calls (requiring as little information as possible from the process environment)
- Generic as possible (different parts of the same package as well as other packages could use the same RPC)

## How to Execute an RPC from a Client Application

1. If your RPC has any input parameters beyond the mandatory first parameter, set a Param node in the TRPCBroker's Param property for each. For each input parameter, set the following sub properties:
  - Value
  - PType (Literal, List, or Reference).

If the parameter's PType is List, however, set a list of values in the Mult subproperty rather than setting the Value subproperty.

Here is an example of some settings of the Param property:

```
RPCBroker1.Param[0].Value := '10/31/97';
RPCBroker1.Param[0].PType := literal;
RPCBroker1.Param[1].Mult['"NAME"'] := 'SMITH, JOHN';
RPCBroker1.Param[1].Mult['"SSN"'] := '123-45-6789';
RPCBroker1.Param[1].PType := list;
```

2. Set the TRPCBroker's RemoteProcedure property to the name of the RPC to execute.

```
RPCBroker1.RemoteProcedure:='A6A LIST';
```

3. Invoke the Call method of the TRPCBroker component to execute the RPC. All calls to the Call method should be done within an exception handler **try...except** statement, so that all communication errors (which trigger the EBrokerError exception) can be trapped and handled. For example:

```
try
    RPCBroker1.Call;
except
    On EBrokerError do
        ShowMessage('A problem was encountered communicating with the
server. ');
end;
```

4. Any results returned by your RPC are returned in the TRPCBroker component's Results property. Depending on how you set up your RPC, results are returned either in a single node of the Results property (Result[0]) or in multiple nodes of the Results property.



You can also use the `lstCall` and `strCall` methods to execute an RPC. The main difference between these methods and the `Call` method is that `lstCall` and `strCall` do not use the Results property, instead returning results into a location you specify.

## RPC Security: How to Register an RPC

Security for RPCs is handled through the RPC registration process. Each client application must create a context for itself, which checks if the application user has access to a "B"-type option in the Kernel menu system. Only RPCs assigned to that option can be run by the client application.

To enable your application to create a context for itself:

1. Create a "B"-type option in the OPTION file (#19) for your application.



The OPTION TYPE "**B**" represents a **B**roker client/server type option.



2. In the RPC multiple for this option type, add an entry for each RPC that your application calls. You can also specify a security key that can lock each RPC (this is a pointer to the SECURITY KEY file) and M code in the RULES subfield that can also determine whether to enable access to each RPC.
3. When you export your package using KIDS, export both your RPCs and your package option.
4. Your application must create a context for itself on the server, which checks access to RPCs. In the initial code of your client application, make a call to the CreateContext method of your TRPCBroker component. Pass your application's "B"-type option's name as a parameter. For example:

```
RPCBroker1.CreateContext(option_name)
```

If the CreateContext method returns True, only those RPCs designated in the RPC multiple of your application option will be permitted to run.

If the CreateContext method returns False, you should terminate your application (if you don't your application will run, but you will get errors every time you try to access an RPC).

5. End-users of your application must have the "B"-type option assigned to them on one of their menus, in order for CreateContext to return True.

### Bypassing RPC Security for Development

Having the XUPROGMODE security key allows you to bypass the Broker security checks. You can run any RPC without regard to application context (without having to use the CreateContext method). This is a convenience for application development. When you complete development, make sure you test your application from an account **without** the XUPROGMODE key, to ensure that all RPCs needed are properly registered.

### BrokerExample Online Code Example

The BrokerExample sample application provided with the BDK demonstrates the basic features of developing RPC Broker-based applications, including:

- Connecting to an M server
- Creating an application context
- Using the GetServerInfo function
- Displaying the **VISTA** splash screen
- Setting the RPCBroker.Param property for each Param PType (literal, reference, list)
- Calling RPCs with the Call method
- Calling RPCs with the lstCall and strCall methods

## Remote Procedure Calls (RPCs)

The client source code files for the BrokerExample application are located in the SAMPLES\BROKEREX subdirectory of the main BDK32 directory.

## 4. Other RPC Broker APIs

### GetServerInfo Function

The GetServerInfo function retrieves the end-user workstation's server and port. Use this function to set the TRPCBroker component's Server and ListenerPort properties to reflect the end-user workstation's settings before connecting to the server.

If there is more than one server/port to choose from, GetServerInfo displays dialog that allows users to select a service to connect to, as shown below:

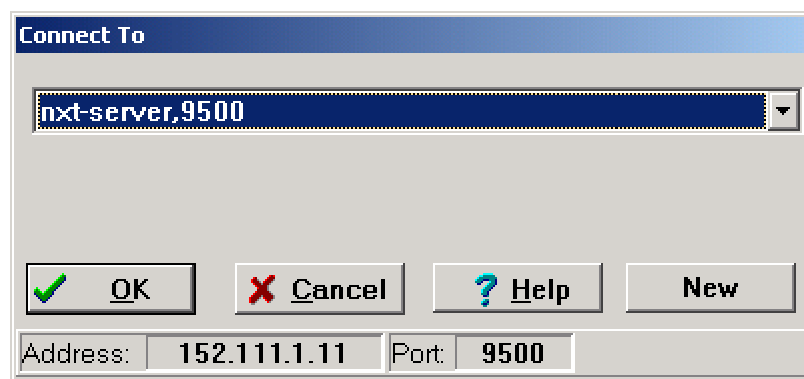


Figure 1: Server and port configuration selection dialog

If exactly one server and port entry is defined in the Microsoft Windows Registry, GetServerInfo does not display this dialog. The values in the single Microsoft Windows Registry entry are returned, with no user interaction required.

If more than one server and port entry exists in the Microsoft Windows Registry, the dialog is displayed, and the user chooses to which server they want to connect.

If no values for server and port are defined in the Microsoft Windows Registry, GetServerInfo does not display this dialog, and automatic default values are returned (i.e., BROKERSEVER and 9200).

#### Syntax of GetServerInfo function:

```
function GetServerInfo(var Server, Port: string): integer;
```



The unit is RpcConfl.

## VISTA Splash Screen Procedures

Two procedures in SplVista.PAS unit are provided to display a **VISTA** splash screen when an application loads:

```
procedure SplashOpen;  
procedure SplashClose(TimeOut: longint);
```

It is recommended that the splash screen be opened and closed in the section of Pascal code in an application's project file (i.e., .DPR).

To use the splash screen in an application:

1. Open your application's project (.DPR) file (in Delphi, choose View | Project Source).
2. Include the SplVista in the uses clause of the project source.
3. Call SplashOpen immediately after the first form of your application is created and call SplashClose just prior to invoking the Application.Run method.
4. Use the TimeOut parameter to ensure a minimum display time.



Figure 2: VISTA Splash screen

```

uses
  Forms, Unit1 in 'Unit1.pas', SplVista;

  {$R *.RES}

begin
  Application.Initialize;
  Application.CreateForm(TForm1, Form1);
  SplashOpen;
  SplashClose(2000);
  Application.Run;
end.

```

## XWB GET VARIABLE VALUE RPC

You can call the XWB GET VARIABLE VALUE RPC (distributed with the RPC Broker) to retrieve the value of any M variable in the server environment. Pass the variable name in Param[0].Value and the type (reference) in Param[0].PType. Also, the current context of your user must give them permission to execute the XWB GET VARIABLE VALUE RPC (it must be included in the RPC multiple of the "B"-type option registered with the CreateContext function).

For example:

```

RPCBroker1.RemoteProcedure := 'XWB GET VARIABLE VALUE';
RPCBroker1.Param[0].Value := 'DUZ';
RPCBroker1.Param[0].PType := reference;
try
  RPCBroker1.Call;
except
  On EBrokerError do
    ShowMessage('Connection to server could not be established!');
end;
ShowMessage('DUZ is '+RPCBroker1.Results[0]);

```

## M Emulation Functions

### Piece Function

The Piece function is a scaled down Pascal version of M's \$PIECE function. It is declared in MFUNSTR.PAS.

```

function Piece(x: string; del: string; piece: integer) : string;

```

### Translate Function

The Translate function is a scaled down Pascal version of M's \$TRANSLATE function. It is declared in MFUNSTR.PAS.

```

function Translate(passedString, identifier, associator: string): string;

```

## Encryption Functions

Kernel and the RPC Broker provide some rudimentary encryption and decryption functions. Data can be encrypted on the client end and decrypted on the server, and vice-versa.

### In Delphi

Include HASH in the "uses" clause of the unit in which you'll be encrypting or decrypting.

Function prototypes are as follows:

```
function Decrypt(EncryptedText: string): string;  
function Encrypt(NormalText: string): string;
```

### On the M Server

To encrypt:

```
>S CIPHER=$$ENCRYP^XUSRB1("Hello world!") W CIPHER  
  
/U'11TG~TV1&f-
```

To decrypt:

```
>S PLAIN=$$DECRYP^XUSRB1(CIPHER) W PLAIN  
  
Hello world!
```

## \$\$BROKER^XWBLIB

Use this function in the M code called by an RPC to determine if the Broker is executing the current process. It returns 1 if this is true, 0 if false.

## \$\$RTRNFMT^XWBLIB

Use this function in the M code called by an RPC to change the return value type that the RPC will return on the fly. This allows you to change the return value type to any valid return value type (SINGLE VALUE, ARRAY, WORD PROCESSING, GLOBAL ARRAY, or GLOBAL INSTANCE). It also lets you set WORD WRAP ON to True or False, on the fly, for the RPC.



For more information about \$\$RTRNFMT^XWBLIB, please refer to the "RPC Broker Developer's Guide" (i.e., BROKER.HLP, online help distributed with the BDK).

## 5. Debugging and Troubleshooting

### How to Debug Your Client Application

Beside the normal debugging facilities provided by Delphi, you can also invoke a debug mode so that you can step through your code on the client side and your RPC code on the M server side simultaneously.

To do this:

1. On the client side, set the DebugMode property on the TRPCBroker component to True. When the TRPCBroker component connects with this property set to True, you will get a dialog indicating your workstation IP address and the port number.
2. At this point, switch over to the M server and set any break points in the routines being called in order to help isolate the problem. Then issue the M debug command (e.g., ZDEBUG in DSM).
3. Start the following M server process:

```
>D EN^XWBTCP
```

You will be prompted for the workstation IP address and the port number. After entering the information, switch over to the client application and click on the OK button.

4. You can now step through the code on your client and simultaneously step through the code on the server side for any RPCs that your client calls.

### RPC Error Trapping

M errors on the server that occur during RPC execution are trapped by the use of M and Kernel error handling. In addition, the M error message is sent back to the Delphi client. Delphi will raise an exception EBrokerError and a popup box displaying the error. At this point RPC execution terminates and the channel is closed.

## Troubleshooting Connections

### Identifying the Listener Process on the Server

On DSM systems, where the Broker Listener is running, the Listener process name is `RPCB_Port:NNNN`, where `NNNN` is the port number being listened to. This should help quickly locate Listener processes when troubleshooting any connection problems.

### Identifying the Handler Process on the Server

On DSM systems the name of a Handler process is `ipXXX.XXX:NNNN`, where `XXX.XXX` are the last two octets of the client IP address and `NNNN` is the port number.

### Testing Your RPC Broker Connection

To test the RPC Broker connection from your workstation to the M Server, use the RPC Broker Diagnostic Program (`RPCTEST.EXE`).



For a complete description of the RPC Broker Diagnostic program, please refer to the Troubleshooting chapter of the RPC Broker Systems Manual.



## 6. RPC Broker Developer Utilities

### Programmer Settings

You can use BrokerProgPref.EXE to define certain default property values for the TRPCBroker component. When you place TRPCBroker component(s) on your form(s) in Delphi, the settings you define are used as the default property values.

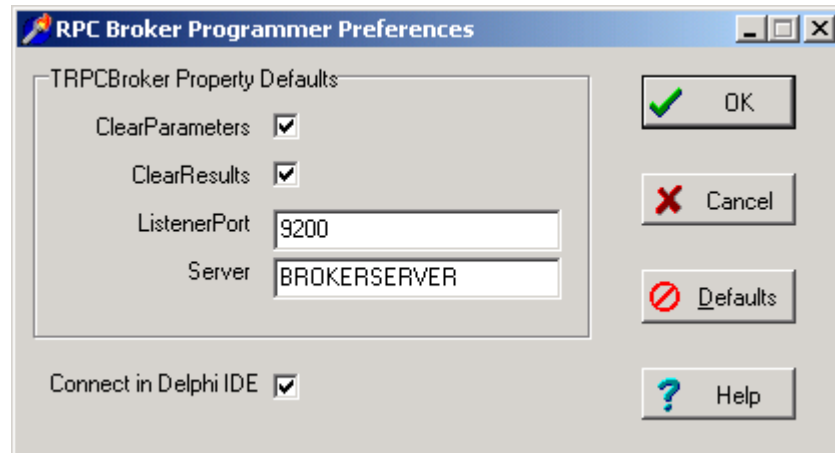


Figure 3: RPC Broker Programmer Preferences dialog

You may want to make an entry for BrokerProgPref.EXE in Delphi's Tools Menu, to make it easily accessible from within Delphi.

Setting	Description
<b>ClearParameters</b>	If checked, sets the ClearParameters property of a TRPCBroker component to True when you add one to a form.
<b>ClearResults</b>	If checked, sets the ClearResults property to of a TRPCBroker component to True when you add one to a form.
<b>ListenerPort</b>	Sets the ListenerPort property of a TRPCBroker component to the specified value when you add one to a form.
<b>Server</b>	Sets the Server property of a TRPCBroker component to the specified value when you add one to a form.
<b>Connect in Delphi IDE</b>	Enables or disables the connection to the M server from within the Borland Integrated Development Environment (IDE). Disabling this is useful when you are developing in an environment without a connection to an M server. For example, when editing certain server properties, an attempt is made to connect to the Server (if enabled).

Table 7: Programmer preference settings



## 7. RPC Broker and Delphi

The following topics highlight changes made to or comments about the Broker to accommodate a particular version of Delphi. They are listed in reverse Delphi version order.

### Delphi 6.0 Packages

#### Delphi V. 6 Standard Edition *Not* Recommended for BDK Development

Delphi V. 6 comes in three flavors: Standard, Professional, and Enterprise. It is recommended that either the Professional or Enterprise version of Delphi 6 be used to develop applications using the RPC Broker.



For more information on the different editions of Delphi, please refer to the "Delphi V. 5 Standard Edition Not Recommended for BDK Development" topic below.

#### XWB\_Rxx.BPL File

The installation of Patch XWB\*1.1\*13 removed the run-time bpl for the Broker package, if it is present in the System32 directory. This run-time package is found in the Delphi6\Projects directory. While on client workstations, the proper location for the run-time bpl is in the System32 directory (so that it is in the system path), however, on developer workstations, this frequently leads to problems when packages are recompiled and the newly generated bpl file is not moved into this directory. To simplify matters on the developer workstation, it is best that the file *not* be put into the System32 directory.

### Delphi 5.0 Packages

#### Delphi V. 5 Standard Edition *Not* Recommended for BDK Development

Delphi V. 5 comes in three flavors: Standard, Professional, and Enterprise. This version of the BDK requires the Professional or Enterprise Edition. Standard edition is targeted mainly at students, and as such leaves out many features. We do *not* recommend using the Standard edition of Delphi V. 5 for RPC Broker development at this time:

1. Delphi V. 5 Standard Edition does not ship with the OpenHelp help system, whose purpose is to allow easy integration of 3rd party component help with Delphi's own internal component help.
2. The RPC Broker component has a dependency on a VCL source code unit, "dsgnintf.pas". Delphi V. 5 Standard Edition does not ship the "dsgnintf" file, in either .PAS or .DCU form. VCL Source code units are available in Delphi 5 Professional and Enterprise Editions. When installing Delphi 5 Professional or Enterprise Edition, make sure you leave the VCL Source installation option selected.

## XWB\_Rxx.BPL File

The installation of Patch XWB\*1.1\*13 removed the run-time bpl for the Broker package, if it is present in the System32 directory. This run-time package is found in the Delphi5\Projects\Bpl directory. While on client workstations, the proper location for the run-time bpl is in the System32 directory (so that it is in the system path), however, on developer workstations, this frequently leads to problems when packages are recompiled and the newly generated bpl file is not moved into this directory. To simplify matters on the developer workstation, it is best that the file *not* be put into the System32 directory.

## Delphi 4.0 Packages

Patch XWB\*1.1\*14 split the Broker package into separate run- and design-time packages. If a package is defined with the VistaBroker.DPK as a Required package, you must delete that required package and add XWB\_Dxx.DPK (where xx=30 for Delphi 3.0, =40 for Delphi 4.0, or =50 for Delphi 5.0) as a required package.

## XWB\_Rxx.BPL File

The installation of Patch XWB\*1.1\*13 removed the run-time bpl for the Broker package, if it is present in the System32 directory. This run-time package is found in the Delphi4\Bin directory. While on client workstations, the proper location for the run-time bpl is in the System32 directory (so that it is in the system path), however, on developer workstations, this frequently leads to problems when packages are recompiled and the newly generated bpl file is not moved into this directory. To simplify matters on the developer workstation, it is best that the file *not* be put into the System32 directory.

## Delphi 3.0 Packages

Delphi V. 3.0 enabled applications to be distributed in specially compiled dynamic-link libraries called Delphi Package Libraries (DPLs). DPLs enable code sharing, reduction of executable file size, and conservation of system resources. The use of DPLs is restricted to development in Delphi 3 or higher.

## VistaBroker.DPL

You can compile the TRPCBroker component code into your application, or you can choose instead to link your application to the VistaBroker.DPL dynamic link library.



For more information on how to use DPLs, see the "Borland Delphi 3 User's Guide."

The VistaBroker.DPL file is installed on end-user workstations by the RPC Broker V. 1.1 end-user client workstation installation program, along with the other RPC Broker V. 1.1. client files. It is installed in an appropriate directory, depending on the operating system (e.g., \WINDOWS\SYSTEM) such that the DPL is accessible when an application calls it. Therefore, you do not need to distribute VistaBroker.DPL with your application.

## Distributing the Delphi VCL30.DPL

If you choose to access functionality of the TRPCBroker component through the VistaBroker.DPL dynamic link library, an additional requirement is that the VCL30.DPL library (provided with Delphi V. 3.0) be installed on the end-user workstation.

The RPC Broker does not distribute VCL30.DPL to end-user workstations (it only distributes VistaBroker.DPL). You must ensure that VCL30.DPL is also installed on end-user workstations, perhaps through the installation instructions you provide to system managers.



## 8. RPC Broker Dynamic Link Library (DLL)

### DLL Interface

The RPC Broker provides a Dynamic Link Library (DLL) interface, which acts like a "shell" around the Delphi TRPCBroker component. The DLL is contained in the file BAPI32.DLL.

The DLL interface enables client applications, written in any language that supports access to Microsoft Windows DLL functions, to take advantage of all features of the TRPCBroker component. This allows programming environments other than Borland Delphi to make use of the TRPCBroker component. All of the communication to the server is handled by the TRPCBroker component, accessed via the DLL interface.

### Exported Functions

The complete list of functions exported in the DLL is provided in the "RPC Broker Developer's Guide" (i.e. BROKER.HLP, online help distributed with the BDK). Functions are provided in the DLL for:

- Creating and destroying TRPCBroker components.
- Setting and retrieving TRPCBroker component properties.
- Executing TRPCBroker component methods.

### Header Files Provided

The following header files provide correct declarations for DLL functions:

C	BAPI32.H
C++	BAPI32.HPP
Visual Basic	BAPI32.BAS

### Sample DLL Application

The Visual Basic (VB) EGCHO sample application distributed with the Broker Development Kit (see the ../BDK32/Samples/Vb5Egcho directory), demonstrates use of the TRPCBroker DLL from Microsoft Visual Basic.

## Return Values from RPCs

Results from an RPC executed on an M server are returned as a text stream. This text stream may or may not have embedded <CR><LF> character combinations.

When you call an RPC using the TRPCBroker component for Delphi, the text stream returned from an RPC is automatically parsed and returned in the TRPCBroker component's Results property as follows:

<b>Results stream contains &lt;CR&gt;&lt;LF&gt; combinations</b>	<b>Location/format of results</b> <i>(assumes RPC's Word Wrap On field is True if RPC is GLOBAL ARRAY or WORD PROCESSING type)</i>
Yes	Results nodes, split based on <CR><LF> delimiter
No	Results[0]

**Table 8: TRPCBroker component's Results property**

When you call an RPC using the DLL interface, the return value is the unprocessed text stream, which may or may not contain <CR><LF> combinations. It is up to you to parse out what would have been individual Results nodes in Delphi, based on the presence of any <CR><LF> character combinations in the text stream.

## COTS Development and the DLL

The Broker DLL serves as the gateway to the REMOTE PROCEDURE file (#8994) for non-Delphi client/server applications. In order to use any RPCs not written specifically by the client application (e.g., CONSULTS FOR A PATIENT, USER SIGN-ON RPCs, or the more generic FileMan RPCs), you must call the RPC Broker DLL with input parameters defined and results accepted in the formats required by the RPC being called.

Therefore, to use the Broker DLL interface you must determine the following information for each RPC you plan to use:

- How does the RPC expect input parameters, if any, to be passed to it?
- Will you be able to create any input arrays expected by the RPC in the same format expected by the RPC?
- What will the results data stream returned by the RPC look like?



## 9. For More Information

### RPC Broker Developer's Guide—BROKER.HLP

This manual provides an overview of development with the RPC Broker.



For more complete information on development with the RPC Broker components, please refer to the "RPC Broker Developer's Guide" (i.e., BROKER.HLP, online help distributed with the BDK).

You may want to make an entry for BROKER.HLP in Delphi's Tools Menu, to make it easily accessible from within Delphi. To do this, use Delphi's Tools | Configure Tools option. Create a new menu entry similar to the following:

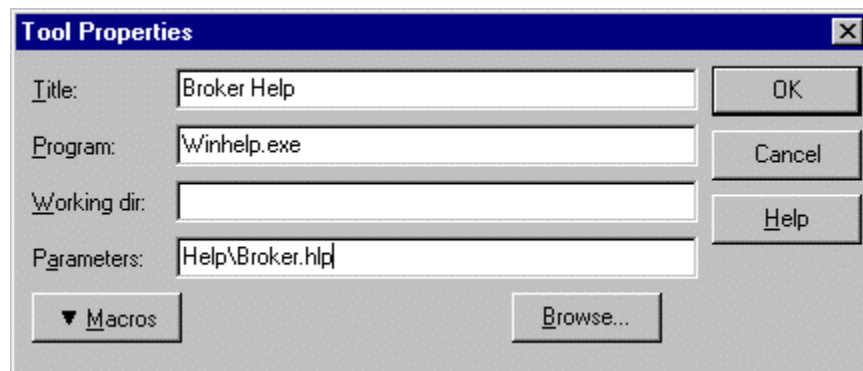


Figure 4: Delphi's Tool Properties dialog

#### BROKER.HLP as Context-sensitive Help Within Delphi

The BROKER.HLP file provides context-sensitive help within Delphi on the TRPCBroker component and its associated properties and methods. This help is available when you have installed the RPC Broker V 1.1 BDK. When installed, you can select the TRPCBroker component or one of its properties in the Object Inspector, and press the F1 key to get help on that item.



For more information on installing the RPC Broker in Delphi, please consult the "RPC Broker Installation Guide."

## Other RPC Broker Resources

To learn more about the RPC Broker, consult these related resources:

Title	Format
Developer's Guide	Microsoft Windows Help (i.e., BROKER.HLP, online help distributed with the BDK)
Release Notes	Adobe Acrobat PDF
Installation Guide	Adobe Acrobat PDF
Systems Manual	Adobe Acrobat PDF
Technical Manual	Adobe Acrobat PDF
Security Guide	Adobe Acrobat PDF

**Table 9: Related documentation (and format)**

## RPC Broker Web Site

Additional information about the RPC Broker, as well as all RPC Broker manuals, is available at the RPC Broker web site:

<http://vista.med.va.gov/broker/>

# Glossary

ACCESS CODE	A code that, along with the Verify code, allows the computer to identify you as a user authorized to gain access to the computer. Your code is greater than 6 and less than 20 characters long; can be numeric, alphabetic, or a combination of both; and is usually assigned by a site manager or application coordinator. It is used by the Kernel's Sign-on/Security system to identify the user (see Verify Code).
ALERTS	Brief online notices that are issued to users as they complete a cycle through the menu system. Alerts are designed to provide interactive notification of pending computing activities, such as the need to reorder supplies or review a patient's clinical test results. Along with the alert message is an indication that the View Alerts common option should be chosen to take further action.
ANSI MUMPS	The MUMPS programming language is a standard recognized by the American National Standard Institute (ANSI). MUMPS stands for <b>M</b> assachusetts <b>U</b> tility <b>M</b> ulti- <b>p</b> rogramming <b>S</b> ystem and is abbreviated as M.
APPLICATION PACKAGE	Software and documentation that support the automation of a service, such as Laboratory or Pharmacy within VA medical centers. The Kernel application package is like an operating system relative to other <b>VISTA</b> applications.
CALLABLE ENTRY POINT	An authorized programmer call that may be used in any <b>VISTA</b> application package. The DBA maintains the list of DBIC-approved entry points.
CARET	A symbol expressed as up caret ("^"), left caret ("<"), or right caret (">"). In many M systems, a right caret is used as a system prompt and an up caret as an exiting tool from an option. Also known as the up-arrow symbol or shift-6 key.
CLIENT	A single term used interchangeably to refer to the user, the workstation, and the portion of the program that runs on the workstation. In an object-oriented environment, a client is a member of a group that uses the services of an unrelated group. If the client is on a local area network (LAN), it can share resources with another computer (server).
COMPONENT	An object-oriented term used to describe the building blocks of GUI applications. A software object that contains data and code. A component may or may not be visible. These components interact with other components on a form to create the GUI user application interface.
COTS	<b>Commercial Off-the-Shelf</b> . COTS refers to software packages that can be purchased by the public and used in support of <b>VISTA</b> .

DATA DICTIONARY	<p>The Data Dictionary is a global containing a description of the kind of data that is stored in the global corresponding to a particular file. VA FileMan uses the data internally for interpreting and processing files.</p> <p>A Data Dictionary (DD) contains the definitions of a file's elements (fields or data attributes), relationships to other files, and structure or design. Users generally review the definitions of a file's elements or data attributes; programmers review the definitions of a file's internal structure.</p>
DBIA	<p><b>Database Integration Agreement</b>, a formal understanding between two or more application packages that describes how data is shared or how packages interact. The DBA maintains a list of DBIAs between package developers, allowing the use of internal entry points or other package-specific features that are not available to the general programming public.</p>
DEFAULT	<p>A response the computer considers the most probable answer to the prompt being given. In the roll-and-scroll mode of <i>VISTA</i>, the default value is identified by double forward slash marks (//) immediately following it. In a GUI-based application the default may be a highlighted button or text. This allows you the option of accepting the default answer or entering your own answer. To accept the default you simply press the enter (or return) key. To change the default answer, type in your response.</p>
DIRECT MODE UTILITY	<p>A programmer call that is made when working in direct programmer mode. A direct mode utility is entered at the M prompt (e.g., &gt;<b>D ^XUP</b>). Calls that are documented as direct mode utilities <i>cannot</i> be used in application package code.</p>
DLL	<p><b>Dynamic Link Library</b>. A DLL allows executable routines to be stored separately as files with a DLL extension. These routines are only loaded when a program calls for them. DLLs provide several advantages:</p> <ol style="list-style-type: none"><li>1. DLLs help save on computer memory, since memory is only consumed when a DLL is loaded. They also save disk space. With static libraries, your application absorbs all the library code into your application so the size of your application is greater. Other applications using the same library will also carry this code around. With the DLL, you don't carry the code itself, you have a pointer to the common library. All applications using it will then share one image.</li><li>2. DLLs ease maintenance tasks. Because the DLL is a separate file, any modifications made to the DLL will not affect the operation of the calling program or any other DLL.</li><li>3. DLLs help avoid redundant routines. They provide generic functions that can be utilized by a variety of programs.</li></ol>

ERROR TRAP	A mechanism to capture system errors and record facts about the computing context such as the local symbol table, last global reference, and routine in use. Operating systems provide tools such as the %ER utility. The Kernel provides a generic error trapping mechanism with use of the ^%ZTER global and ^XTER* routines. Errors can be trapped and, when possible, the user is returned to the menu system.												
FORUM	The central e-mail system within <b>VISTA</b> . Developers use FORUM to communicate at a national level about programming and other issues. FORUM is located at the Washington, DC CIO Field Office (162-2).												
GUI	<b>Graphical User Interface</b> . A type of display format that enables users to choose commands, initiate programs, and other options by selecting pictorial representations (icons) via a mouse or a keyboard.												
ICON	A picture or symbol that graphically represents an object or a concept.												
IRM	<b>Information Resource Management</b> . A service at VA medical centers responsible for computer management and system security.												
KERNEL	A set of <b>VISTA</b> software routines that function as an intermediary between the host operating system and the <b>VISTA</b> application packages (e.g., Laboratory, Pharmacy, IFCAP, etc.). Kernel provides a standard and consistent user and programmer interface between application packages and the underlying M implementation. (VA FileMan and MailMan are self-contained to the extent that they can standalone as verified packages.) Some of Kernel's components are listed below along with their associated namespace assignments:												
	<table> <tr> <td>KIDS</td><td>XPD</td></tr> <tr> <td>Menu Management</td><td>XQ</td></tr> <tr> <td>Tools</td><td>XT</td></tr> <tr> <td>Sign-on/Security</td><td>XU</td></tr> <tr> <td>Device Handling</td><td>ZIS</td></tr> <tr> <td>Task Management</td><td>ZTM</td></tr> </table>	KIDS	XPD	Menu Management	XQ	Tools	XT	Sign-on/Security	XU	Device Handling	ZIS	Task Management	ZTM
KIDS	XPD												
Menu Management	XQ												
Tools	XT												
Sign-on/Security	XU												
Device Handling	ZIS												
Task Management	ZTM												
MENU MANAGER	The Kernel module that controls the presentation of user activities such as menu choices or options. Information about each user's menu choices is stored in the Compiled Menu System, the ^XUTL global, for easy and efficient access.												
MULTIPLE	A multiple-valued field; a subfile. In many respects, a multiple is structured like a file.												
MUMPS (ANSI STANDARD)	A programming language recognized by the American National Standards Institute (ANSI). The acronym MUMPS stands for <b>Massachusetts General Hospital Utility Multi-programming System</b> and is abbreviated as M.												

NAMESPACING	A convention for naming <b>VISTA</b> package elements. The Database Administrator (DBA) assigns unique character strings for package developers to use in naming routines, options, and other package elements so that packages may coexist. The DBA also assigns a separate range of file numbers to each package.
NODE	In a tree structure, a point at which subordinate items of data originate. An M array element is characterized by a name and a unique subscript. Thus the terms: node, array element, and subscripted variable are synonymous. In a global array, each node might have specific fields or "pieces" reserved for data attributes such as name.
OPTION	As an item on a menu, an option provides an opportunity for users to select it, thereby invoking the associated computing activity. In <b>VISTA</b> , an entry in the OPTION file (#19). Options may also be scheduled to run in the background, non-interactively, by TaskMan.
PROMPT	The computer interacts with the user by issuing questions called <i>prompts</i> , to which the user returns a response.
REMOTE PROCEDURE CALL	A remote procedure call (RPC) is essentially M code that may take optional parameters to do some work and then return either a single value or an array back to the client application.
ROUTINE	A program or a sequence of instructions called by a program that may have some general or frequent use. M routines are groups of program lines that are saved, loaded, and called as a single unit via a specific name.
SECURITY KEY	The purpose of Security Keys is to set a layer of protection on the range of computing capabilities available with a particular software package. The availability of options is based on the level of system access granted to each user.
SERVER	The computer where the data and the Business Rules reside. It makes resources available to client workstations on the network. In <b>VISTA</b> , it is an entry in the OPTION file (#19). An automated mail protocol that is activated by sending a message to a server at another location with the "S.server" syntax. A server's activity is specified in the OPTION file (#19) and can be the running of a routine or the placement of data into a file.
SIGN-ON/SECURITY	The Kernel module that regulates access to the menu system. It performs a number of checks to determine whether access can be permitted at a particular time. A log of signons is maintained.
SUBSCRIPT	A symbol that is associated with the name of a set to identify a particular subset or element. In M, a numeric or string value that: is enclosed in parentheses, is appended to the name of a local or global variable, and identifies a specific node within an array.

UCI	User Class Identification, a computing area. The MGR UCI is typically the Manager's account, while VAH or ROU may be Production accounts.
USER ACCESS	<p>This term is used to refer to a limited level of access to a computer system that is sufficient for using/operating a package, but does not allow programming, modification to data dictionaries, or other operations that require programmer access. Any of <i>VISTA</i>'s options can be locked with a security key (e.g., XUPROGMODE, which means that invoking that option requires programmer access).</p> <p>The user's access level determines the degree of computer use and the types of computer programs available. The Systems Manager assigns the user an access level.</p>
USER INTERFACE	The way the package is presented to the user, such as Graphical User Interfaces that display option prompts, help messages, and menu choices. A standard user interface can be achieved by using Borland's Delphi Graphical User Interface to display the various menu option choices, commands, etc.
VERIFY CODE	The Kernel's Sign-on/Security system uses the Verify code to validate the user's identity. This is an additional security precaution used in conjunction with the Access code. Verify codes shall be at least eight characters in length and contain three of the following four kinds of characters: letters (lower- and uppercase), numbers, and, characters that are neither letters nor numbers (e.g., "#", "@" or "\$"). If entered incorrectly, the system does not allow the user to access the computer. To protect the user, both codes are invisible on the terminal screen.
VISTA	Veterans Health Information Systems and Technology Architecture. <i>VISTA</i> includes the VA's application software (i.e., Microsoft Windows-based and locally-developed applications, roll-and-scroll, and interfaces such as software links to commercial packages). In addition, it encompasses the VA's uses of new automated technology including the clinical workstations. <i>VISTA</i> encompasses the rich automated environment already present at local VA medical facilities.
WINDOW	An object on the screen (dialog) that presents information such as a document or message.





## Appendix A—Patch Revision History

The following table displays the patch/version release history for the RPC Broker software. The sequence number (Seq #) is the order in which the patch was released by National **VISTA** Support (NVS) and installed by the site. The sequence number does not necessarily match the Patch ID number in all cases. Also, the sequence number, in some cases, can imply dependency between patches. Each table entry indicates that the documentation was reviewed and updated as needed for each patch; in some cases, a patch may not affect the content of the documentation. Regardless, the patch will still be added to the patch history in reverse patch sequence order.

Seq #	Patch ID	Brief Summary	Status
22	XWB*1.1*26	<p>This patch updates the Broker's Programmer Client Workstation software—also known as the Broker Development Kit (BDK). It supports Delphi V. 4, 5, and 6.</p> <p>It provides a SharedRPCBroker component. Any GUI application that uses the SharedRPCBroker will now have the ability to share a Broker connection. This patch also supports ESSO.</p>	<p>Client-side only patch—Not yet released.</p> <p>This document has been reviewed and updated as needed for this patch.</p>
21	XWB*1.1*13	<p>This patch updates the Broker's Programmer Client Workstation software—also known as the Broker Development Kit (BDK). It supports Delphi V. 4, 5, and 6.</p> <p>It provides Silent Login functionality in the Broker. Any GUI RPC Broker-based application will now have the ability to login to an M Server silently (i.e. without any user dialog). This patch also supports Enterprise Single-Sign-On (ESSO).</p>	<p>Client and server patch—Not yet released.</p> <p>This document has been reviewed and updated as needed for this patch.</p>
20	XWB*1.1*27	<p>This patch enables asynchronous processing, multiple jobs running at the same time. Prior to this patch, processing of requests to the HL7 package for remote data made by GCPR and CPRS, was performed synchronously - in order of time of request, each job finishing before the next job started.</p>	<p>Server-side only patch—Patch released on 03/15/02.</p>
19	XWB*1.1*16	<p>This patch provides several bug fixes (e.g., READ/WRITE errors) initiated via NOIS.</p>	<p>Server-side only patch—Patch released on 02/06/02.</p>

Seq #	Patch ID	Brief Summary	Status
18	XWB*1.1*24	<p>This patch updates the Broker's Programmer Client Workstation software—also known as the Broker Development Kit (BDK). It supports only Delphi V. 4 and Delphi V. 5.</p> <p>Due to version-dependent code, a problem was recently encountered that is associated with reading the Microsoft Windows Registry in programs compiled with Delphi V. 5. Because a conditional test was specifically looking for Delphi V. 4-based applications, Delphi V. 5-based applications ended up using Broker code for Delphi V. 3. This can result in users having limited privileges, preventing their ability to read data from the registry. This has been observed when a user with limited NT privileges attempts to select a location for the RPC Broker connection, and it results in the use of the default BrokerServer/9200. However, users with higher levels of NT access do not see this problem. This version-dependent code was removed via this patch.</p>	<p>Client and server patch—Patch released on 11/09/01.</p> <p>This document was reviewed and updated as needed for this patch.</p>
17	XWB*1.1*22	<p>The calling site had a NEW PERSON file entry with a phone number containing a trailing backslash ("\"). As part of Remote Data Views (RDV), this data was then encoded and sent to the remote site.</p> <p>At the remote site, a bug caused the backslash ("\") to be appended to the end of several other strings, which then caused the reported error. This was fixed by correcting the decoding routine.</p> <p>Because the error occurred before RDV was setup to handle an error, it caused the calling site to keep sending the same message repeatedly. This has been fixed by setting an error trap at the beginning of RDV.</p> <p>If the application does not set some data into the return variable, XWB2HL7 will return a string starting with "-1^".</p> <p>The XWB EXAMPLE option, RPC's and routine (XWBEXMPL) are included to add an entry point for testing that will record the symbol table in the error trap.</p>	<p>Server-side only patch—Patch released on 10/03/01.</p> <p>This document was reviewed and updated as needed for this patch.</p>

Seq #	Patch ID	Brief Summary	Status
16	XWB*1.1*20	<p>This patch addresses the following:</p> <ul style="list-style-type: none"> <li>During the early testing of RDV (Remote Data View), the DUZ value was hard set to .5 just before the call to the RPC. This was done because the code to set up the user at the remote site wasn't ready. When the code was fixed to properly set the DUZ, the old code was never removed. This has been fixed in the routine XWB2HL7.</li> <li>If data was left in the ^XUTL("XQ",\$J,"IO")node it could cause problems when HOME^%ZIS is called by some RPC's, so this ^XUTL node is killed off before the RPC is called.</li> <li>In an e-mail message from CPRS developers: The global that may be used to pass data back to the RPC was not killed before its use. This was fixed in the routine XWBDRPC.</li> </ul>	<p>Server-side only patch—Patch released on 05/10/01.</p> <p>This document was reviewed and updated as needed for this patch.</p>
15	XWB*1.1*14	<p>This patch updates the Broker's Programmer Client Workstation software—also known as the Broker Development Kit (BDK). It adds no new functionality . It does the following:</p> <ul style="list-style-type: none"> <li>Releases the source code for the BDK.</li> <li>Splits the VistaBroker package into separate design- and run-time packages.</li> </ul>	<p>Client and server patch—Patch released on 10/17/00.</p> <p>This document was reviewed and updated as needed for this patch.</p>
14	XWB*1.1*18	<p>This patch fixed the following NOIS: LOM-0800-62301 and PRO-0800-11778:</p> <p>If there are problems associated with the remote site's HL7 definitions—specifically the receiving application. Then the RPC XWB REMOTE STATUS CHECK will get an UNDEF error on the variable Z.</p>	<p>Server-side only patch—Patch released on 10/17/00.</p> <p>This document was reviewed and updated as needed for this patch.</p>
13	XWB*1.1*12	<p>This patch is in support of the CPRS Remote Data Views project. The RPC Broker is used to facilitate invocation of Remote Procedure calls on a remote server. The RPC Broker uses <b>VISTA</b> HL7 as the vehicle to pass RPC name and parameters from a local server to a remote server. On the return path, <b>VISTA</b> HL7 is also used to send results from the remote server back to the local server.</p>	<p>Server-side only patch—Patch released on 08/04/00.</p> <p>This document was reviewed and updated as needed for this patch.</p>

Seq #	Patch ID	Brief Summary	Status
12	XWB*1.1*10	<p>This patch gives greater information about and control of RPCs. Specific new abilities are:</p> <ul style="list-style-type: none"> <li>Blocking an RPC either locally*, remotely*, or in both contexts by setting a value in the INACTIVE field of the Remote Procedure file. Prior to this patch, values in this field had no effect.</li> <li>Assuring that an RPC is at least a specified version when it is run remotely* by setting a value in the new VERSION field of the REMOTE PROCEDURE file.</li> <li>Querying a server regarding the status of RPCs by using new Remote Procedures: XWB IS RPC AVAILABLE and XWB ARE RPCS AVAILABLE.</li> <li>In addition, this patch stops M errors from occurring when a client application attempts to: <ul style="list-style-type: none"> <li>1.) Create a context that does not exist on the server, or</li> <li>2.) Run a remote procedure that does not exist on the server.</li> </ul> </li> </ul>	<p>Server-side only patch—Patch released on 08/04/00.</p> <p>This document was reviewed and updated as needed for this patch.</p>
11	XWB*1.1*15	<p>This patch should correct a problem on Cache sites with the Broker looping with COMMAND errors. This error is caused when the Broker tries to open the TCP port and the port is already open via the Broker.</p>	<p>Server-side only patch—Patch released on 04/12/00.</p> <p>This document was reviewed and updated as needed for this patch.</p>
10	XWB*1.1*11	<p>This patch updates the Broker's Programmer Client Workstation software—also known as the Broker Development Kit (BDK)—adding support for Delphi V. 5 development.</p>	<p>Client and server patch—Patch released on 01/24/00.</p> <p>This document was reviewed and updated as needed for this patch.</p>

Seq #	Patch ID	Brief Summary	Status
9	XWB*1.1*9	<p>This patch fixes the following:</p> <ul style="list-style-type: none"> <li>Intersystems License. This is the patch that works with Patch XU*8*118. The code to share licenses when GUI and Telnet users from the same workstation are connected is in place and ZU now calls it. This patch adds a similar call from XWBTCP.</li> <li>This patch brings a new XWB LISTENER STOP ALL option for shutting down multiple listeners. It also brings a modified option XWB LISTENER STARTER for starting Broker listeners.</li> </ul>	<p>Server-side only patch—Patch released on 01/24/00.</p> <p>This document was reviewed and updated as needed for this patch.</p>
8	XWB*1.1*8	<p>This patch supports GUI Multi-Divisional signon. If a user has more than one division to choose from, the user must select one before continuing with the signon. If the user has only one division in File #200, this division will be used; otherwise, the default institution in the KERNEL SYSTEM PARAMETERS file will be used.</p>	<p>Client-side only patch—Patch released on 12/10/99.</p> <p>This document was reviewed and updated as needed for this patch.</p>
7	XWB*1.1*6	<p>This patch does the following:</p> <ul style="list-style-type: none"> <li>Eliminates server Broker jobs for which there is no client application.</li> <li>Changes the time that the server waits for the client to contact it. A new field in the KERNEL SYSTEM PARAMETERS file, BROKER ACTIVITY TIMEOUT (default value of approximately 3 minutes) controls the length of the timeout.</li> </ul>	<p>Client and server patch—Patch released on 09/09/99.</p> <p>This document was reviewed and updated as needed for this patch.</p>
6	XWB*1.1*4	<p>This patch does the following:</p> <ol style="list-style-type: none"> <li>Introduces a shorter timeout when logging in via any GUI RPC Broker-based application. The server listener process will timeout after 90 seconds if the user has not passed in his/her Access and Verify codes.</li> <li>Updates the Broker's Programmer Client Workstation software—also known as the Broker Development Kit (BDK)—adding support for Delphi V. 4 development.</li> <li>Fixes a bug in which the Title bar of the Kernel Login form was being changed when a user started entering their Access code.</li> </ol>	<p>Client and server patch—Patch released on 06/24/99.</p> <p>This document was reviewed and updated as needed for this patch.</p>

Seq #	Patch ID	Brief Summary	Status
5	XWB*1.1*7	<p>This patch addresses two problems:</p> <ol style="list-style-type: none"> <li>1. A command error is occurring at RESTART+17^XWBTCP when the Broker tries to reopen a device that is not closed. This seems to be a problem with Cache sites only. The result of this error causes the Broker Listener to stop. The fix is in XWBTCP.</li> <li>2. The listener doesn't check for available slots before starting a new process. The listener will now check the MAX SIGNON ALLOWED field of the VOLUME SET multiple in the KERNEL SYSTEM PARAMETERS file, the same one used by Kernel logon. This fix is also in XWBTCP.</li> </ol>	<p>Server-side only patch—Patch released on 06/04/99.</p> <p>This document was reviewed and updated as needed for this patch.</p>
4	XWB*1.1*5	<p>This patch is for the support of RUM. This will allow the trapping of data for Remote Procedure Calls (RPCs) and the RPC Broker handler.</p>	<p>Server-side only patch—Patch released on 03/31/99.</p> <p>This document was reviewed and updated as needed for this patch.</p>
3	XWB*1.1*3	<p>Under CPRS, when the DG routines call OP^XQCHK to record what option is used, it was getting back "unknown." The Broker created context needed to set the variable XQY.</p>	<p>Server-side only patch—Patch released on 01/06/99.</p> <p>This document was reviewed and updated as needed for this patch.</p>
2	XWB*1.1*2	<p>This patch addresses three problems with RPC Broker v1.1:</p> <ul style="list-style-type: none"> <li>• Encrypted Literal—Pattern match failure in RPCs. The failure only occurs with RPCs that combine multiple literals and an array (NOIS WAS-0398-22800).</li> <li>• Data Collection Switch turned "Off"—Collection of data will be controlled by the use of the Capacity Management tools (NOIS BRX-0498-11768 and HUN-0498-21137).</li> <li>• 10 Second Network Timeout in Client Agent—A 30 second timeout is being switched to 10 for network communications with the Client Agent.</li> </ul>	<p>Server-side only patch—Patch released on 07/27/98.</p> <p>This document was reviewed and updated as needed for this patch.</p>

Seq #	Patch ID	Brief Summary	Status
<b>1</b>	XWB*1.1*1	<p>This patch fixes some small problems that were discovered after release (server-side only).</p> <ul style="list-style-type: none"> <li>• XWBTCP—Remove the SYMBOL_TABLE from the VAX DSM JOB command.</li> <li>• XWBTCP—When stopping the Broker, see a failure to open a socket.</li> <li>• XWB BROKER EXAMPLE option—This option was missing its type field.</li> </ul>	<p>Server-side only patch—Patch released on 02/18/98.</p> <p>This document was reviewed and updated as needed for this patch.</p>
<b>NA</b>	Version 1.1	Original Version 1.1 software release.	September 1997

**Table 10: RPC Broker V. 1.1 patch revision history (in reverse sequence order)**





# Index

## A

About this Version of the BDK, 1-1  
Appendix A—Patch Revision History, 1  
Assumptions About the Reader, xi

## B

Backward Compatibility Issues, 1-1  
BAPI32.DLL, 8-1  
BROKER.HLP, 9-1  
BROKER^XWBLIB, 4-4  
BrokerExample, 3-8  
BrokerProgPref.EXE, 6-1  
Bypassing Security for Development, 3-7

## C

Call method, 3-6  
Call Method, 2-3  
Calls  
    Discrete, 3-5  
    Silent, 3-5  
Commonly Used Terms, x  
Compatibility Issues, 1-1  
Connect To, 4-1  
Connection  
    Testing Your RPC Broker Connection, 5-2  
Contents, v  
Context -sensitive Help, Delphi V. 6.0, 7-1  
Context-sensitive Help, 9-1  
COTS Development and the DLL, 8-2  
Create Your Own RPCs, 3-1  
CreateContext method, 3-7, 4-3  
CreateContext Method, 2-3

## D

Data Dictionary  
    Data Dictionary Utilities Menu, xi  
    Listings, xi  
Debugging, 5-1–5-2  
Debugging and Troubleshooting, 5-1  
DECRYP^XUSRB1, 4-4  
Decrypt Method, 4-4  
Delphi  
    3.0 Packages, 7-3  
    4.0 Packages, 7-2

5.0 Packages, 7-2

6.0 Packages, 7-1

Delphi and RPC Broker, 7-1

Developer Utilities, 6-1

Developer's Guide, Online, 9-1

Diagnostic Program, 5-2

Discrete Calls, 3-5

DLL Interface, 8-1

Documentation History, iii

Documentation Symbols, ix

DPL, 7-3

Dynamic Link Library (DLL), 8-1–8-2

## E

EN^XWBTCIP, 5-1

ENCRYP^XUSRB1, 4-4

Encrypt Method, 4-4

Encryption/Decryption Functions, 4-4

Error Message Handling, 5-1

Execute an RPC from a Client Application, How  
to, 3-6

Exported Functions, 8-1

## F

Figures, vii

First Input Parameter (Required), 3-2

For More Information, 9-1

## G

GetServerInfo Method, 2-2, 2-4, 4-1

## H

Header Files, 8-1

Help

    At Prompts, x

    Online, x

Help in Delphi V. 6.0, 7-1

Home Pages

    Adobe Acrobat Quick Guide Web Address, xii

    Adobe Systems Incorporated Web Address,  
    xii

    RPC Broker Home Page Web Address, 9-2

    RPC Broker Web Address, xii

    SD&D Home Page Web Address, xi

How to

- Connect to an M Server, 2-4
- Debug Your Client Application, 5-1
- Execute an RPC from a Client Application, 3-6
- Generate Technical Information Online, x
- Register an RPC, 3-7
- How To
  - Use this Manual, ix

## I

- Identifying
  - Handler Process on the Server, 5-2
  - Listener Process on the Server, 5-2
- Information, 9-1
- Input Parameter Types (Optional), 3-4
- Introduction, 1-1
- Issues
  - Backward Compatibility, 1-1

## L

- List File Attributes Option, xi
- lstCall Method, 2-3

## M

- M Emulation Functions, 4-3
- Menus
  - Data Dictionary Utilities, xi
- Message Handling, Errors, 5-1
- Methods and Properties
  - TRPCBroker Component, 2-1
- MFUNSTR.PAS, 4-3
- Microsoft Windows Registry, 2-4, 4-1

## O

- Online
  - Documentation, x
  - Help Frames, xi
  - Technical Information, How to Generate, x
- Online Code Samples (RPCs), 3-8
- OPTION file (#19), 3-7
- Options
  - List File Attributes, xi
- Orientation, ix
- Other RPC Broker APIs, 4-1
- Other RPC Broker Resources, 9-2

## P

- Patch History, 1
- Piece Function, 4-3

- Programmer Settings, 6-1
- Properties and Methods
  - TRPCBroker Component, 2-1

## Q

- Question Mark Help, x

## R

- Reader, Assumptions About the, xi
- Reference Materials, xii
- Registering RPCs, 3-7
- Registry, 2-4, 4-1
- Relationship Between an M Entry Point and an RPC, 3-1
- Remote Procedure Calls (RPCs), 3-1–3-8
  - Bypassing Security, 3-7
  - Executing, 3-6
  - M Entry Points, 3-2–3-4
  - Online Code Samples, 3-8
  - Registering, 3-7
  - RPC Entry, 3-5
- REMOTE PROCEDURE File, 3-1, 3-5, 8-2
- Return Value Types, 3-2
- Return Values from RPCs, 8-2
- Revision History
  - Documentation, iii
  - Patches, 1
- RPC Broker and Delphi, 7-1
- RPC Entry in the Remote Procedure File, 3-5
- RPC Error Trapping, 5-1
- RPC M Entry Point Examples, 3-4
- RPC Security
  - How to Register an RPC, 3-7
- RPCTEST.EXE, 5-2
- RTRNFMT^XWBLIB, 4-5

## S

- Sample DLL Application, 8-1
- Silent Calls, 3-5
- Splash Screen, 4-2
- SplashClose Method, 4-2
- SplashOpen Method, 4-2
- SplVista.PAS Unit, 4-2
- Standard Edition, 7-1, 7-2
- strCall Method, 2-3
- Symbols Found in the Documentation, ix
- Syntax of GetServerInfo Function, 4-1

**T**

- Terms, Commonly Used, x
- Testing Your RPC Broker Connection, 5-2
- Translate Function, 4-4
- Trapping RPC Errors, 5-1
- Troubleshooting, 5-1–5-2
- Troubleshooting and Debugging, 5-1
- Troubleshooting Connections, 5-2
- TRPCBroker component
  - CreateContext method, 3-7
  - Key Properties, 2-2
- TRPCBroker Component, 2-1–2-4
  - Call method, 3-6
  - Call Method, 2-3
  - Connecting to an M Server, 2-4
  - CreateContext Method, 2-3, 4-3
  - IstCall Method, 2-3
  - Methods, 2-3
  - Properties and Methods, 2-1
  - strCall Method, 2-3

**U**

- URLs
  - Adobe Acrobat Quick Guide Web Address, xii
  - Adobe Systems Incorporated Web Address, xii
  - RPC Broker Home Page Web Address, 9-2
  - RPC Broker Web Address, xii
  - SD&D Home Page Web Address, xi
- Use this Manual, How to, ix
- Utilities, 6-1

**V**

- VBEGCHO, 8-1
- Version
  - About this Version of the BDK, 1-1
- Vista Splash Screen, 4-2
- VistaBroker.DPL, 7-3
- Visual Basic, 8-1

**W**

- Web Pages
  - Adobe Acrobat Quick Guide Web Address, xii
  - Adobe Systems Incorporated Web Address, xii
  - RPC Broker Home Page Web Address, 9-2
  - RPC Broker Web Address, xii
  - SD&D Home Page Web Address, xi
- Web Site
  - RPC Broker Home Page, 9-2
- What is a Remote Procedure Call?, 3-1
- What Makes a Good Remote Procedure Call?, 3-5
- Windows Registry, 2-4, 4-1
- Writing M Entry Points for RPCs, 3-2

**X**

- XUPROGMODE Security Key, 3-7
- XWB GET VARIABLE VALUE RPC, 4-3
- XWB\_Rxx.BPL File, 7-1, 7-2
- XWBLIB
  - \$\$BROKER^XWBLIB, 4-4
  - \$\$RTRNFMT^XWBLIB, 4-5

